Claude Code Workflow Setup Guide

Context Engineering for Productive AI-Assisted Development

Prepared by: Mehul Jariwala
Lead Full Stack + AI/ML Engineer

Version 1.0 | February 2026

Table of Contents

1. Overview

This guide sets up a structured Claude Code workflow based on **context engineering** — the practice of feeding the right context at the right time to get predictable, high-quality AI output.

Why This Workflow?

- **Never exceed 60% context** — quality degrades sharply beyond this threshold
- **Phase-based execution** — Research → Plan → Implement → Validate, clearing context between each
- **Persistent artifacts** — all research, plans, and reports saved to a thoughts/ directory that survives context clears
- **Reusable by the entire team** — custom commands and CLAUDE.md files work for anyone on the project

> **Proven Result:** 72 story points completed in a single sprint (React refactoring, bug fixes, new features). The .03 release shipped stable with no major issues.

2. Prerequisites

- **Claude Code** installed globally: `npm install -g @anthropic-ai/claude-code`
- **Node.js** (v18+) and **npm** installed
- A project repository where you want to set up the workflow
- (Optional) **HumanLayer** for the thoughts directory tooling: `npx humanlayer thoughts init`

3. Step 1: Set Up the thoughts/ Directory

The `thoughts/` directory is the persistent memory layer. Research, plans, and validation reports are saved here so they survive context clears and can be referenced across sessions and by the entire team.

Option A: Automatic Setup

```
npx humanlayer thoughts init
```

Option B: Manual Setup

Run these commands in your project root:

```
mkdir -p thoughts/{shared,searchable}
mkdir -p thoughts/shared/{research,plans,prs}
```

```
mkdir -p thoughts/<your-name>/{tickets,notes}
```

> **Tip:** The `searchable/` folder should contain hard links to all documents so Claude's search tools can find relevant context quickly. This directory can be sym-linked globally across repos so the entire team and org share knowledge.

4. Step 2: Create Custom Commands

Create the commands directory in your project root:

```
mkdir -p .claude/commands
```

Then create the four command files below. Each file defines an agent role that Claude Code will assume when you invoke the command.

5. Command: /research_codebase

File: `.claude/commands/research_codebase.md`

Usage: `/research_codebase + {What I want to research or build}`

Output: `thoughts/shared/research/YYYY-MM-DD_.md`

File Contents:

```
You are a research agent. Your job is to understand the existing codebase related to the user's
query.

1. Spawn parallel sub-agents to:
- Locate all relevant files in the codebase
- Analyze how the current implementation works
- Search thoughts/ directory for existing research
- Extract insights from any relevant documents

2. Return findings with exact file:line references

3. Save output to: thoughts/shared/research/YYYY-MM-DD_.md

Include:
- Summary of findings
- Code references with file:line numbers
- Architecture insights
- Open questions

DO NOT write any code. Research only.
```

Notes: Spawns parallel agents to investigate different aspects simultaneously. Produces a comprehensive research document with exact file:line references. No code is written during this phase.

6. Command: /create_plan

File: `.claude/commands/create_plan.md`

Usage: `/create_plan @thoughts/shared/research/.md`

Output: `thoughts/shared/plans/YYYY-MM-DD-.md`

File Contents:

```
You are a planning agent. Read the provided research document and create a detailed
implementation plan.

1. Read the research document thoroughly
2. Ask clarifying questions before drafting
3. Create an iterative plan with:
- Clear phases with specific changes
- Exact file paths to modify
- Code snippets showing what to add/change
- Automated verification (tests, linting, type checking)
- Manual verification checklist
- Success criteria for each phase

4. Save to: thoughts/shared/plans/YYYY-MM-DD-.md

Expect 5+ iterations. First draft is never final. Keep refining until every phase is actionable.
```

Notes: This is interactive — iterate at least 5 times. The plan is the source of truth. Each phase should have automated verification commands and manual test checklists. Budget 30-45 minutes for planning; it saves hours during implementation.

7. Command: /implement_plan

File: `.claude/commands/implement_plan.md`

Usage: `/implement_plan @thoughts/shared/plans/.md Phase 1 only`

Output: `Code changes + verification results`

File Contents:

```
You are an implementation agent. Execute the plan ONE PHASE AT A TIME.

1. Read the full plan document
2. Read all files mentioned in the current phase
3. Make ONLY the changes specified for this phase
4. Run all automated verification:
- Unit tests
- Linting
- Type checking
5. Report results and pause for manual testing

DO NOT proceed to the next phase until the current phase passes all checks and the user confirms
manual testing is complete.
```

Notes: Critical rule: ONE phase at a time. After automated checks pass, Claude pauses and waits for manual testing confirmation before proceeding. If context exceeds 60%, clear and restart with the plan file for the next phase.

8. Command: /validate_plan

File: `.claude/commands/validate_plan.md`

Usage: `/validate_plan @thoughts/shared/plans/.md`

Output: `Validation report`

File Contents:

```
You are a validation agent. Systematically verify the implementation against the plan.

1. Read the plan document
2. Check recent git commits
3. Run all automated verification commands
4. Review code changes against plan specifications
5. Generate a validation report:
- Correctly implemented items
- Deviations from plan
- Issues needing fixes
- Manual testing checklist
```

Notes: This is your safety net. If gaps are found, loop back to /implement_plan for the missing items. The validation report catches deviations, missing edge cases, and items that automated tests can't cover.

9. Step 3: The 4-Phase Workflow

For every feature, bug fix, or refactoring task, follow this cycle:

Phase 1: Research

```
/research_codebase How does X work in our codebase?
```

→ Check /context → if >60%, /clear

Phase 2: Plan

```
/create_plan @thoughts/shared/research/YYYY-MM-DD_topic.md
```

→ Iterate 5+ times → /clear

Phase 3: Implement

```
/implement_plan @thoughts/shared/plans/YYYY-MM-DD-topic.md Phase 1 only
```

→ Test → /clear → repeat for Phase 2, 3...

Phase 4: Validate

```
/validate_scenarios @thoughts/shared/plans/YYYY-MM-DD-topic.md
```

→ Fix any gaps → loop back to implement if needed

10. Context Management Rules

- **Never exceed 60% context capacity.** Check with /context during sessions. Quality and predictability degrade sharply beyond this.
- **Clear context after each major phase.** Each phase starts clean with only the artifacts it needs from the previous phase.
- **Save everything to thoughts/ directory.** Research findings, plans, and validation reports are persisted as files, not kept in memory.
- **Reference files instead of keeping in memory.** Attach the output file from the previous phase rather than relying on conversation history.
- **Use parallel agents to gather information efficiently.** The research phase spawns sub-agents to investigate different aspects simultaneously.

11. Directory Structure Reference

```
thoughts/
■■■ <your-name>/ # Personal notes and tickets
```

```
■ ■■■ tickets/
■ ■■■ notes/
■■■ shared/ # Team-shared documents
■ ■■■ research/ # Research output files
■ ■■■ plans/ # Implementation plans
■ ■■■ prs/ # PR-related documentation
■■■ searchable/ # Hard links for Claude search

.claude/
■■■ commands/
■ ■■■ research_codebase.md
■ ■■■ create_plan.md
■ ■■■ implement_plan.md
■ ■■■ validate_plan.md
■■■ CLAUDE.md # Project-specific instructions
```

12. Quick Start Checklist

- ☐ **1.** Install Claude Code: `npm install -g @anthropic-ai/claude-code`
- ☐ **2.** Create thoughts/ directory (manual or `npx humanlayer thoughts init`)
- ☐ **3.** Create `.claude/commands/` directory with the 4 command files
- ☐ **4.** Create `CLAUDE.md` with your project-specific standards (design system, patterns, linting rules)
- ☐ **5.** Pick a small feature and run the full 4-phase cycle
- ☐ **6.** After first feature — reflect: Where did you get stuck? Did you maintain <60% context? Iterate on the workflow.

About CLAUDE.md

Create a `CLAUDE.md` (or `.claude/CLAUDE.md`) file in your project root. This file is automatically loaded by Claude Code and should contain:

- Project-specific coding standards and conventions
- Design system tokens (colors, spacing, typography) for your project
- Component library patterns and template conventions
- Linting rules and test patterns
- Architecture decisions and patterns to follow
- Any domain-specific terminology or business rules

Team Multiplier: Once CLAUDE.md is configured, any team member using Claude Code on the same repo will automatically get the same standards-compliant output. This is reusable infrastructure — set it up once, benefit across the entire team.